mysqljsonimport 1.6

JSON file importer for MySQL

mysqljsonimport 1.6
2013-05-13
Anders Karlsson
anders@papablues.com

Table of contents

TABLE OF CONTENTS **2**

INTRODUCTION **2**

BUILDING MYSQLJSONIMPORT **3**

CONFIGURING AND USING MYSQLJSONIMPORT **4**

INITIALIZING PROCESSING 5
Global initialization *5*
Thread level initialization *5*
UTF-8 support *6*
TESTING INITIALIZATION 6
JSON FILE FORMATS 6
JSON Row object format *6*
JSON Array format *6*
JSON Object format *6*
STATUS PRINTOUT 7

MYSQLJSONIMPORT CONFIGURATION OPTIONS REFERENCE **7**

CHANGE LOG **9**

VERSION 1.6 9
VERSION 1.5 9
VERSION 1.4 9
VERSION 1.3 9
VERSION 1.2 10
VERSION 1.1 10
VERSION 1.0 10

Introduction

MySQL has come a long way and when this document is written, in May 2012, MySQL 5.6 is around the corner. At the same time, there is a lot of competition, the different NoSQL alternatives are breaking new ground where MySQL used to be king. In reality, what has opened up is not only a market with several alternatives that are much less common than what used to be the case (i.e. “Which SQL database should I choose”) but an environment where these alternatives coexist.

In addition to this, there is now also many more data formats, partly because of these new technologies (which many, me included, doesn’t really consider “new” in terms of not having existed before), and partly because we deal with new kinds of data, like GPS positioning data, document based data, lists of items, objects etc.

One of the many new formats of data that is appearing is JSON (JavaScript Object Notation). JSON has proved to be very popular not only with the JavaScript focused technologies, but as a generic format used in conjunction with REST and as a data transformation format.

In the case of MySQL, there isn't really that much JSON support. Instead, MySQL seems to rely on CSV (Coma Separated Values) for data exchange. CSV is in some ways a neat format, but it is not standardized and has it's roots in table/column/field only world. We now have objects, we have to understand that, and we have unstructured data. This fits easily in JSON and MySQL can sure handle it, but there are few, or any tools to support this. So along comes **mysqljsonimport**. This is a tool that allows you to import a file with JSON based data to be imported into MySQL. It is not a very advanced tool, but at least it understands JSON and MySQL and does some basic mappings, with the intention that data is then massaged further by standard SQL.

The program is a command line based tool, and relies on *jansson* for JSON processing. Beyond that, the importing process is multi-threaded and supports MySQL array inserts, and has a bunch of configuration options, which may be used in a configuration file, on the command line, or a combination of the two.

Building mysqljsonimport

mysqljsonimport use the *autobuild* tools for building, so building it follows these steps:

- Download **mysqljsonimport** package from sourceforge: https://sourceforge.net/projects/mysqljson
- Unpack it
- Configure it
- Make it
- Optionally install it

Like this example:

```
$ wget http://sourceforge.net/projects/mysqljson/files/myjsonimport_1.6/mysqljsonimport
      -1.6.tar.gz
$ tar xvfz mysqljsonimport-1.6.tar.gz
$ cd mysqljsonimport-1.6
$ ./configure –with-mysql=/usr/local/mysql
$ make
```

Following this, the program is available in the directory where you built it. You can install it to the location specified with the optional *--prefix* (the installation will copy the program to the *bin* subdirectory of this location) option to *configure* (the default is */usr*). If you install to /usr you need root privileges of course.

```
$ sudo make install
```

Once the program is built, we are ready to configure it. You can either use a normal MySQL configuration file, pass the options on a command line, or use a combination of the two. For normal MySQL connection options, like socket, port and username, these are also read from the *client* section

MySQL usual configuration files. See mysqljsonimport configuration options reference for a reference to all options.

If you have problems building mysqljsonimport, the most likely issues are:

- The MySQL libraries aren't found. MySQL must be installed and if not in any of the standard places, then pass the *--with-mysql* option.
- The *Jansson* library isn't found. This library is used for JSON parsing and is needed by *mysqljson*. Look at the Jansson website (http://www.digip.org/jansson) for instructions of where to download it and how to build it.

If you install the *jansson* library in a non-standard location, for example if you do not have privileges to install it into */usr/* or any of the other standard locations, then use the *–with-jansson* option when configuring, to point to the appropriate location. The same is true for MySQL. For example if you installed the jansson library in */home/ken/jansson*, and with MySQL installed in */usr/local/mysql565*, then the configure command would look like this:

```
$ ./configure --with-mysql=/usr/local/mysql565 --with-jansson=/home/ken/jansson
$ make
```

After building mysqlimport, you may optionally test it. The tests are on the *tests* subdirectory, and are run like this:

```
$ make check
```

And this will run all tests. As the tests connect to MySQL, you might have to adjust the MySQL connection setting for this to work, and these are in the *tests/test-connect.cnf* file. Still, the occationoal test may fail also, as not all tests rely on this config file, but most should work.

Configuring and using mysqljsonimport

The default mysqljsonimport configuration file is *mysqljson.cnf* and the options are in the *jsonimport* section in this file. The only required option is a database name. If no table to import is specified, then all tables for that database, in the directory with the same name as that database, is loaded. All files in the database directory that has a name that corresponds to the name scheme <table name><extension>, where the default extension is *.json*. You may specify a specific file to import, but then you may only import one table. You may specify one or more specific tables to import, either using the *--table* option, the *table* configuration file option or just passing table names on the commandline.

One you have the configuration file set up, you can start the program, and you probably want it running in the background:

```
£ ./mysqljsonimport --database test --table mydata mydata
```

The way the loading works is that the columns in the table that is imported into determines which attributes of the JSON objects to load. This comparison is case insensitive. A simple example table would be this:

```
CREATE TABLE users(id INTEGER NOT NULL PRIMARY KEY,
  firstname VARCHAR(256) NOT NULL,
```

```
  middlename VARCHAR(256),
  lastname VARCHAR(256) NOT NULL,
  user_group INT NOT NULL,
  active BOOL NOT NULL);
```

If we then have a JSON file, *users.json*, with the following contents:

```
{"_id": 1, "firstname": "John", "middlename": "Frank", "lastname": "Doe", "interests":
      ["cars", "books"], "active": false}
{"_id": 2, "firstname": "Anne", "middlename": null, "lastname": "Doe", "interests":
      ["diving"], "active": true}
```

We have three possible issues:

- The *id* column isn't present in the JSON file, instead this column is called *_id*.
- The value for the *group* column isn't present in the JSON file, but it is a NOT NULL column, so we need to provide a value. We know the group is 57.
- The *interests* column is a JSON array and isn't present.

To fix this, we have to map the column name for the *id* column. The second issue may be ignored, as the *interests* column doesn't exist in the table, it will not be loaded. We may decide to load it into a new column, but then we will have the array as loaded as a string into that column. If we decide to ignore the *interests* column, the command line might look like this:

```
mysqljsonimport --col-value=group=57 --col-json-name=id=_id --table=users --database=test users
```

If you want to use a configuration file instead, then this would look like this:

```
[jsonimport]
database=test
table=users
col-value=group=57
col-name-map=_id=id
file=users.json
```

Initializing processing

When load JSON data, there might be SQL statement that needs to be run before the load itself, and in some cases before the load for each thread.

Global initialization

This might be SQL statements to create tables to be loaded etc. The option to use is *--sql-init* and you may specify multiple such options.

```
sql-init=CREATE TABLE emp(empno INT NOT NULL PRIMARY KEY, ename VARCHAR(256))
sql-init=CREATE INDEX emp_ename ON emp(ename)
```

Thread level initialization

SQL statements specified with the *--sql-thread-init* option are executed for each thread just after it has been started and before any rows are processed. Useful values are to set up appropriate binlogging etc.

UTF-8 support

This is also initialized for each connection and by default UTF-8 is enabled (this is reasonable as JSON strings are always UTF8). If you want to disable UTF8, do not use Thread level initialization for this, but instead use the *--skip-utf8* option.

Testing initialization

To test the init options, you can use the *--dry-run* option which will not load any data, but will run all initializations scripts.

JSON file formats

There are quite a few formats you can find for JSON data. And as of mysqljsonimport version 1.5, three different formats are supported.

JSON Row object format

This was the only format supported in mysqljsonimport before version 1.5, and this is still the default format. In this format, the file contains a number of JSON objects, each representing a row in a table, and the members are mapped to column names in the table to be loaded.

```
{"firstname": "Bill", "lastname": "Clinton"}
{"firstname": "Richard", "middlename": "Millhouse", "lastname": "Nixon"}
```

The advantage of this format is that each row is handled as a separate JSON object, so the file can be loaded one row at the time.

JSON Array format

This format is similar to the row object format, but the objects are members of an array instead of being handled on at the time. The disadvantage to this is that as the array is a JSON object, we have to read and parse the whole array to determine if it is valid JSON. The advantage is that the whole file is one big JSON object, so we can actually use the whole file as such. As we can see, this has both advantages and disadvantages.

It is also supported to have more than one array, like this:

```
[
  {"firstname": "Bill", "lastname": "Clinton"},
  {"firstname": "Richard", "middlename": "Millhouse", "lastname": "Nixon"}
]
[
  {"firstname": "George", "lastname": "Bush"}
]
```

This format is automatically detected, but if for some reason you don't want this, you can use the *--skip-array-file* option.

JSON Object format

This format was added in mysqljsonimport 1.6, and in this case, the data is more object oriented. The file contains one or more arrays of objects, each containing one or more members that is either an object or an array of objects. Each such object then represents a row in a table, with the name of the object representing the table to be loaded. To use this format you have to specific the *--object-format* option. An example is:

```
[
  {"states": {"name": "California", "short": "CA"}
  },
  {"presidents": [
    {"firstname": "Bill", "lastname": "Clinton"},
    {"firstname": "Richard", "middlename": "Millhouse", "lastname": "Nixon"}
    ]
  }
]
```

Status printout

To show status of what is going on while the program is running, send a *sigusr1* signal to the process running it, for example using the Linux *kill* command. Use the ps command to find the process to send it to. For example, with mysqlimport running in the background:

```
$ ps -u someuser | grep mysqlimport
12123 pts/8    00:28:58 mysqljsonimport
$ kill -sigusr1 12123
Loading table mydata, 23 threads running, 349106 rows read, 240000 rows inserted
  349105 rows processed, 30905 rows max per thread
  21 seconds running, 11428 rows processed / s
```

mysqljsonimport configuration options reference

The mysqljsonimport program takes many options, and is very flexible. The options may either be in a configuration file or passed on the command line or both, the options then prefixed with double dashes (--). Some options have single character shortcuts, then they are prefixed with a single dash. When used in the configuration file, only the long option may be used.

In the configuration file, defined by the *–defaults-file* option, the options must be in the *jsonimport* section. The *client* section in the normal MySQL configuration files is also read, to pick to defaults for the MySQL host, socket and such things. Options with a * may be specified more than once.

- array-strip – Skip leading and trailing brackets when loading an array into a column.
- array-as-null – Handle an array in the data being loaded as NULL, the default is to load it as a string.
- skip-array-file - By default, a top level array is broken down and each element is loaded into the database as a separate row, this option stops this.
- bool-as-int – Handle a JSON bool as an int instead of the usual MySQL TRUE / FALSE values.
- skip-col – Skip loading the specified column *
- col-default – Specify column default values. If a column to import isn't included in the data being imported, then it is instead imported with this value. The default is to set undefined columns to NULL. The format of this option is <column name>=<value> *
- col-incr - Increment of an integer column specified with either the col-default or the col-value options. Each occurrence of the column will be increment with the specified value, the default is 0 *
- col-null - Set the specified column to NULL in MySQL *

- col-value – Set a named column in the database to a fixed value, you may have more than one of these options. The format is <column name>=<value> (i.e. *--col-value=col1=57*) *
- col-json-name – Set the name of a MySQL column to a different name in the json file. This option may be specified several times, and has the format <name>=>alternate name>. For example *–col-name-map=id=_id **
- commit-interval – The number of .records per commit in a thread. This is only checked when an array is to be inserted, so more than this # of records may be inserted. If set to 0 or 1, autocommit is used
- database / d – Name of database to use. This is a required option
- defaults-file – Name of configuration file to use
- directory - Directory to load json files from, the default is the same as the name of the database
- dry-run - Do not load any data, just test. Note that init-scripts are run.
- empty-null – Treat an empty string, array or object as NULL
- extension - File extension for json files, the default is *.json*. Note the the dot as part of this string.
- file – Name of file to import *
- help – Show help. This option can only be entered on the commandline.
- host / h – The hostname of the MySQL server. Default is NULL (localhost).
- include - Name of extra configuration file to include *
- insert-array-size – Number of records per array to insert.
- logfile – File for log messages. Default is standard error.
- loglevel – Level of logging. Valid values are status, error, info, verbose or debug. The default level is info.
- max-insert-threads - Max number of parallel insert threads to run for each table
- null-string – An alternative way of representing NULL JOSN values in MySQL, the default is to map a JSON null to a MySQL NULL.
- object-to-null – Load embedded objects in JSON as NULL to the database
- object-format - Load data in JSON object format.
- skip-parallel - Do not do any parallel processing of tables or parts of tables
- password / p – Password for user to connect to the database with
- port / P – MySQL Server port to connect to
- prtrec – Print records as they are loaded
- skip-lines – Skip the specified number of lines before starting to load data
- skip-json-records – Skip the specified number of JSON records before starting to load
- socket / S – The MySQL Server socket to connect to
- sql-init – SQL statements to run before starting to load *
- sql-retry – The number of times to retry a failed MySQL INSERT statement
- sql-retry-sleep – The number of seconds to sleep between retrying an SQL statement
- sql-thread-init - SQL statements to run before processing starts in each individual thread *

- stats - Level of statistics to print after processing. Is one of *none*, *normal* or *full*, where *normal* is the default.
- stop-on-error – Stop if there is an error when executing a clean-sql statement. This is the default
- stop-on-init-error – Stop if one of the sql-init statements fails
- table – Name of the table to import into *
- threading-threshold - Size of a JSON file to enable parallel loading of the file, specified in Mb. The default is 10
- user / u – MySQL user to connect to the database with.
- skip-utf8 – Disable UTF-8 mode.
- version – Show version. This is a command-line only option.

Change log

Version 1.6

This release adds one important feature, and as part of implementing this, a lot of code was restructured. Also this release adds some fixes to the build scripts.

- Added JSON object format load.
- Added --dry-run option.

Version 1.5

This release adds both features and some bugfixing.

- Added JSON array format load.

Version 1.4

This is mainly a bugfix release.

- Added status printout on interrupt.
- Added -h option (shortcut for --host).
- Fixed status summary when program was interrupted.
- Fixed so that the program could be stopped properly.
- Fixed so that an error in loading didn't cause a hang.

Version 1.3

This is a major release with many changes, options and bug fixes.

- Name changed to mysqlimport instead of mysqlload
- Introduced multi table load.
- Introduced load from files in a specific directory (this is in line with the *mysqljsonexport* tool)
- Aligned options to work as they do in *mysqljsonexport*.

- Added dynamic threading, where as many threads as needed are created, up to a specified maximum.

Version 1.2

This version introduces features to make mysqljsonload compatible with mysqljsonunload. This in turn means that there is a slight difference in how mysqljsonload works comared to previous versions. The major change is that mysqljsonload now supports multiple files, and this affects related options. In addition, some features has been added, and some bugs has been fixed.

- Support for loading several files in parallel.
- Support for load directories.
- Support for ignoring foreign key checks.

Version 1.1

This is mainly a feature release, which also has a few bugfixes.

- Fixed bug in the way commit and autocommit is handled.
- Added retry of failed SQL statements.
- Logging is improved with some thread specific data being printed.
- Added skip on leading JSON records, in addition to skipping lines.
- Added handling or true, false and null JSON values.
- Added sql-init and supporting stop-on-init-error options.

Version 1.0

This is the first public release.